Titanium regression test

Table of Contents

Overview . . . 1
Code smell markers . . . 1
  General tests . . . 1
  Code smells . . . 1
Metrics . . . 2

Overview

These tests are responsible for verifying the following functionalities in the titanium plugin:

- calculating metrics,
- signing code smells

This is generally achieved with some carefully written ttcn3 module files, that are known exactly what results we expect when a specific titanium functionality is executed on them. The project `Regression_test_project` is a set of such files. During the test this project is loaded to the workspace, and several test suites are executed against it.

Code smell markers

Code smell markers are tested in two larger test suites: in "General tests" (class `GeneralTests`) and in "Code smells" (class `CodeSmellChecker`).

General tests

In this test code smells are tested in general, regardless any specific code smell. The first part is "noChangeConsistency", where titanium analyzer is executed twice in a row with no changes to the files between. Here we expect that the code smell markers created during the analyses are the same (same type and same location). The second test is "touchChangeConsistency", where titanium analyzer is executed again twice, but before the second run each file in the project gets touched, so the titan semantic analyzer processes these files as they were changed. The expectations are the same as previously: there should be no differences in the code smell markers created during the two analyzes.

Code smells

This test suite is responsible to each code smell separately. There are separate file(s) for each code smell, containing ttcn3 codes snippets that are expected to be marked as code smell, and also snippets to catch false positive warnings (valid code that should not be marked). The tests check if

the specific type of code smell markers shows up there, and only there, where we expect it. Loosely, in pseudocode:

```
foreach t type of code smell:
  set warning level of t to ERROR
  execute analyzer on requested modules;
  check if markers of t are where we expected;
```

Metrics

Metrics are checked in a similar way as code smells: there are ttcn3 files to test each metric separately, with exactly known values for a specific metric. During the test the metrics are executed on the given modules, and check if the results are as expected.

A notable subtlety is that for metrics on entities that are smaller than a module (function, test case, altstep) the test file should contain at most one such entity due to the limitation in the `ModuleMetricsWrapper` class.